Interactive Orthogonal Graph Drawing: Algorithms and Bounds

Ulrich Fößmeier

Universität Tübingen, Wilhelm-Schickard-Institut, Sand 13, 72076 Tübingen, Germany, email: foessmei@informatik.uni-tuebingen.de

Abstract. Incremental graph drawing is a model gaining more and more importance in many applications. We present algorithms that allow insertions of new vertices into an existing drawing without changing the position of the objects drawn so far. We prove bounds for the quality of our drawings and considerably improve on previous bounds. Here the number of bends and the used area are our quality measures. Besides we discuss lower bounds for this problem.

1 Introduction

Representing objects and the relations between them is a task arising permanently in many applications of Computer Science. Therefore the concept of *graphs* is widely used to represent the data. A graph is a set V of *vertices* together with a set $E \subseteq V \times V$ of *edges*. Examples for the use of graphs are Petri nets, entity-relationship diagrams, production planning or data flow diagrams in Software Engineering (see e.g. [4,3]). For human processing it is necessary to *visualize* graphs; sometimes this visualization (called *graph drawing*) is even the desired output of some computation, e.g. in VLSI design [15]. In a graph drawing the vertices of the graph are represented by geometric objects (points, circles, squares, ...) and the edges by simple curves connecting them. The drawing must fulfill certain properties which depend on the specific application. Typical properties are: The edges must be straight lines, they must not cross each other (planar drawing) or they must consist only of horizontal and vertical segments (orthogonal drawing). Moreover the drawing should optimize some cost function; this could be to minimize the used area, to minimize the number of crossings or bends, to minimize the total edge length etc. A survey of graph drawing algorithms and applications can be found in [6].

In many cases not the total amount of data is given in advance, but the vertices of the graph are added one by one and after every insertion a drawing is required. This situation is called *incremental drawing*. Thus incremental drawing has its own importance, moreover it is an important step to the more general concept of *interactive drawing*, where also deletions of vertices are allowed. We investigate incremental drawings using the model of orthogonal drawings: Here graphs with a maximum vertex degree of at most four are drawn in such a way that the

vertices are placed on grid points of an integer grid and the edges run along grid lines. If the drawings are destinated for human observers, inserting a new vertex should not change too much the 'old' drawing. In [11] the idea of the user's 'mental map' is introduced (the way how a human user 'understands' a drawing); adding a new vertex should not destroy the user's mental map. Thus an incremental drawing system must guarantee a) to maintain the user's mental map and b) to handle a single insertion in a reasonable time (we require constant time per insertion).
If adding a new vertex must not change at all the position of the objects drawn so far, we speak of the No-Change-Scenario (the No-Change-Scenario is the best way to support the maintenance of a mental map). Papakostas and Tollis [13] introduced this model and also investigated another one: The Relative-Coordinates-Scenario, where it is allowed to shift every drawn vertex by a constant number of units along the x- and y-axes when inserting a new vertex. They gave first results on the number of bends and the used area in both scenaria and presented some examples.
Obviously the Relative-Coordinates-Scenario is more powerful than the No-Change-Scenario. All drawings in the No-Change-Scenario maintain the conditions for the Relative-Coordinates-Scenario. Hence bounds for the No-Change-Scenario-drawings also hold for the other model.
In this paper we focus on the No-Change-Scenario; we refine the methods of [13] and considerably improve on their bounds for the number of bends and the used area; our algorithm even leads to better results (in general) than the Relative-Coordinates-drawings of Papakostas and Tollis.
The rest of the paper is organized as follows: In Section 2 we give some definitions and describe the basic algorithm. In Section 3 we introduce our technique how we bound the value of some cost function (number of bends), and in the following two sections we present new methods how to improve on these bounds. Section 6 is devoted to another important cost function (used area) and in Section 7 we discuss lower bounds for the problem. Finally Section 8 illustrates our algorithms with some examples.

2 First Results

Papakostas and Tollis [13] gave a first algorithm that computes drawings in the No-Change-Scenario and proved $\frac{8}{3} \cdot n \approx 2.66 \cdot n$ (n being the number of vertices drawn so far) as an upper bound for the number of bends produced by their algorithm. In [12] statistics about the practical behaviour of this algorithm are given. We present a more general approach that can be used to minimize any cost function and analyse it for the example of the number of bends being the cost criterion.
We start with some definitions:
A graph with a maximum vertex degree of at most four is called a *4-graph*.
The *local degree ld(v)* of a vertex v is the number of incident edges at the moment when v is inserted.

A vertex v which is inserted at step t is called *new* at this time; in all further steps $t' > t$ it will be called *old.*
An old vertex v has a *free direction* to the top, if there is no other vertex v' having the same x-coordinate as v and a larger y-coordinate than v **and** if there is no vertical edge segment above of v in the same column (this definition implies that there is no edge being incident to v at its top side). Free directions to the bottom, left and right are defined analogously.
A half straight line starting at an old vertex v and pointing into a free direction of v is called a *free line* (of v).
A vertex v is called *critical* if it has exactly one free direction and *semi-critical* if it has exactly two free directions.
The *bounding box* of a drawing is the smallest rectangle containing the whole drawing.
The four gridlines with the minimum distance to the bounding box without intersecting or touching it are called *basic gridlines.*

To prove the correctness of our algorithms we use the following

Incremental Invariant: Every vertex v of degree *deg(v)* (*deg(v)* ≤ 4) in the current drawing has 4 - *deg(v)* free directions.

Lemma 1. *An algorithm that maintains the incremental invariant can compute an orthogonal drawing of any 4-graph.*

Proof. Let v be a vertex that has to be added to the drawing and $v_1, \ldots, v_{ld(v)}$ its neighbours in the current drawing. The incremental invariant implies that we can draw straight lines starting at $v_1, \ldots, v_{ld(v)}$ that do not cross forbidden objects (other vertices, parallel edges); after leaving the bounding box these lines may have some bends in order to join at a geometric point where we place v. $\Diamond$

The strategy of our first algorithm is the following: Place a new vertex v at one of the following places: Either at the insertion of two basic gridlines ('at the corner' of the bounding box), or at the intersection of a basic gridline with a free line of v' (v' being a neighbour of v). In this case v and v' will be connected by a straight line. Since v has at most four neighbours and every neighbour has at most four free directions, there are at most 20 possible places for v. Also the bends along the new edges have to be placed at these positions. Fig. 1a) shows the possible places for a new vertex v having the neighbours v_1, v_2, v_3.
Our strategy is to choose the place that optimizes the main criterion (e.g. the number of bends) *for this insertion.* Fig. 1b) shows two different possibilities, the first one minimizing the number of bends and the second one minimizing edge crossings.
In [7] we give a simple proof for the following

Lemma 2. *An algorithm following the strategy described above maintains the incremental invariant.*

Next we analyse the behaviour of the algorithm with respect to the number of produced bends.

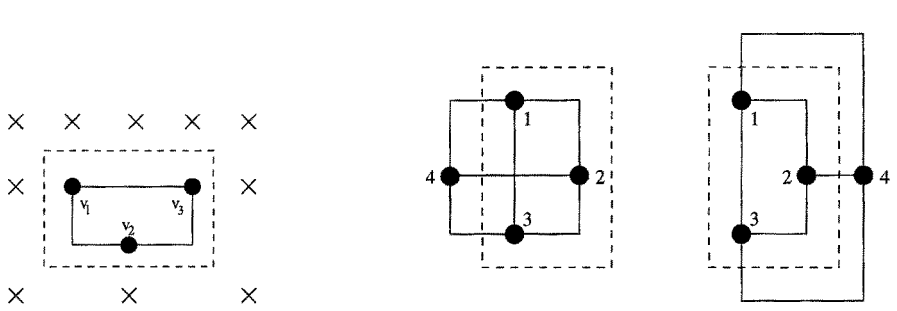

Fig. 1. a) The possible places for vertices and bends b) Minimizing bends or crossings

3 Bounds on the Number of Bends

We subdivide the vertices into different classes according to their local degree and to the number of bends they produce when they are inserted.
B_{ij} denotes the class of vertices with local degree i producing j new bends and $b_{ij} = |B_{ij}|$.
In [13] the graph is demanded to be *connected at any time*, since otherwise the bounds could not be proved. We do not want to restrict our algorithm more than necessary, so we allow insertions with local degree 0; such an insertion (called *ld0-insertion*) creates a new connected component of the graph (which can join with other connected components in the course of the algorithm). Surprisingly, although an insertion of local degree 0 does not cause any bends, the upper bound on the number of bends increases compared to the model where the graph is always connected. This observation encouraged us to distinguish between two classes of problems: With and without ld0-insertions. The analysis of this section forbids vertices with local degree 0 in order to have an easy and understandable proof for the bound.

We distinguish the disjoint classes B_{48}, B_{47}, B_{46}, B_{34}, B_{33}, B_{23}, B_{22}, B_{10}, and characterize the most important classes (B_{46}, B_{33} and B_{22} mean that *at most* 6, 3 respectively 2 bends are needed): Obviously a vertex v with local degree one can always be inserted without any new bends.

The first line of Fig. 2 shows the three different cases how a new vertex v of local degree two can be connected to its neighbours: The two new edges can leave the bounding box at the same side, at different but not opposite sides or at opposite sides (all other situations can be obtained by rotating and reflecting one of the pictures in Fig. 2). According to these cases such an insertion requires one, two or three bends. Thus a vertex $v \in B_{23}$ has two neighbours requiring the incident edges to run in opposite directions; this implies that both neighbours must be critical, otherwise one of the other (cheaper) cases would work.

In the lower part of Fig. 2 the different cases for insertions with local degree three and four can be seen.

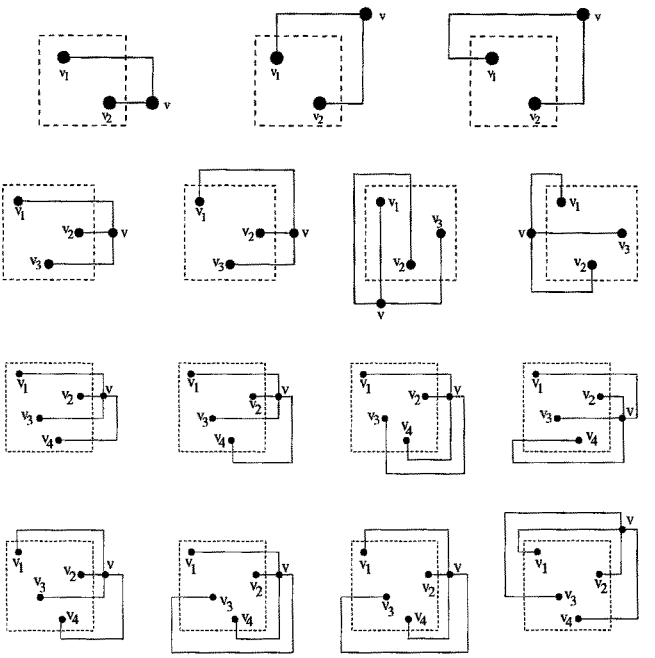

Fig. 2. Insertions of a vertex with local degree two and more

Table 1 shows necessary conditions what kind of neighbours a vertex v must have in order to (possibly) belong to a class B_{ij}.

Table 1

Class	B_{48}	B_{47}	B_{34}	B_{23}
critical neighbours	4	2	1	2

We shortly prove this fact for the example of the class B_{34} (the proof for the other classes is similar): Let v_1, v_2 and v_3 be the neighbours of v; if none of them is critical then there is a pair of them (say v_1 and v_2) having a common free direction; in this case the edge connecting v and v_3 cannot be forced to leave the bounding box in the opposite direction, since v_3 is not critical, and thus the conditions for case B_{34} cannot be fulfilled.

In the following we try to construct a worst case (i.e. a case with a maximum number of bends) obeying the rules of Table 1. This case gives an upper bound for the number of bends needed by our algorithm (in [12] a similar technique was used to prove upper bounds). We use linear programming and maximize the number of bends under certain restrictions. A first Linear Program (LP) could be

$$max \quad 8 \cdot b_{48} + 7 \cdot b_{47} + 6 \cdot b_{46} + 4 \cdot b_{34} + 3 \cdot b_{33} + 3 \cdot b_{23} + 2 \cdot b_{22}$$

$$due\ to\ \ b_{48} + b_{47} + b_{46} + b_{34} + b_{33} + b_{23} + b_{22} + b_{10} = n$$

n is the number of vertices inserted so far. Since the solution of this LP does not give a non-trivial result we must add some more restrictive constraints:

- We evaluate the data from Table 1 and take into account that only vertices that are inserted with a local degree of at most three can ever be or become critical.
- We bound the total number of edges (which is equal to the sum of all local degrees) by $2 \cdot n$.

This leads to the following constraints:

$$crit \leq b_{34} + b_{33} + b_{23} + b_{22} + b_{10}$$

$$4 \cdot b_{48} + 2 \cdot b_{47} + b_{34} + 2 \cdot b_{23} \leq crit$$

$$4 \cdot b_{48} + 4 \cdot b_{47} + 4 \cdot b_{46} + 3 \cdot b_{34} + 3 \cdot b_{33} + 2 \cdot b_{23} + 2 \cdot b_{22} + b_{10} \leq 2 \cdot n$$

A solution of this LP gives $2.5 \cdot n$ as the value of the objective function with the variables $b_{23} = b_{22} = 0.5 \cdot n$, $crit = n$.

As a consequence of all facts discussed in this section we state

Theorem 1. *Our algorithm incrementally computes orthogonal drawings of any 4-graph without ld0-insertions producing at most* $2.5 \cdot n$ *bends (where n is the number of vertices inserted so far); every insertion can be done in constant time.*

Fig. 3 a) shows an easy example for a drawing that exactly realizes this bound: Vertices 1, 2, 3, 4 are already drawn. Vertex 5 has to be connected with 1 and 3, vertex 6 with 2 and 4, vertex 7 with 1 and 4 and vertex 8 with 2 and 3. In the resulting drawing the vertices 5 and 7 play the role of the vertices 1 and 2 and the vertices 6 and 8 play the role of the vertices 3 and 4 and thus we can iterate.

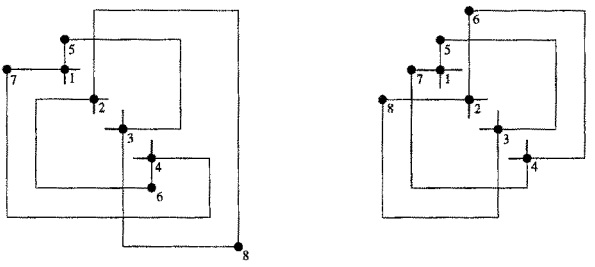

Fig. 3. a) An example with $2.5 \cdot n$ bends b) A drawing with less bends

4 A First Improvement

In the drawing of Fig. 3 a) there are many insertions of type B_{23} using critical vertices having their free direction at opposite sides. If all critical vertices would have the same shape this bad example could not arise. Thus our strategy is to produce always the same type of critical and semi-critical vertices, if possible. In the example of Fig. 3 a) this means that the insertions of type B_{22} (e.g. 5 and 6) create *the same* type of critical vertices, say vertices being free to the left or to the bottom. Following this strategy the vertices 5, 6, 7 and 8 could be inserted with 8 bends (instead of 10 bends) and the resulting drawing has four semi-critical vertices of the same shape, thus making following insertions even cheaper. Following the iteration the graph can be drawn with n bends instead of $2.5 \cdot n$ bends (see Fig. 3 b)).

This effect can also be proved theoretically: We take into account that e.g. an insertion of type B_{23} not only needs two critical vertices but two critical vertices having the free direction at opposite sides. Thus in the LP two types of B_{23} insertions are distinguished. One of them consumes a critical vertex with free direction to the left and a critical vertex with free direction to the right, and the other one consumes each a critical vertex being free to the top and to the bottom. Both types of insertion create a semi-critical vertex. Here it is a free choice of the algorithm whether this semi-critical vertex is free to a) the left and to the top, b) the top and to the right, c) the right and to the bottom or d) the bottom and to the left. If e.g. the two critical neighbours of the new vertex u are free to the left (v_1) resp. to the right (v_2), the algorithm can realize case a) by placing u in the left upper corner of the bounding box, case b) by placing u at the right upper corner of the bounding box etc.
In the LP we establish a variable for every type of insertion (e.g. new vertex with local degree two, both neighbours are critical, one of them being free to the left, the other to the right), and an equality for every type of vertex (e.g. critical being free to the right). These equalities ensure that the same number of vertices of some type is consumed and created. The resulting LP consists of 16 equalities and 3073 variables and can be constructed automatically (in order to avoid mistakes that are inevitable when dealing with so many data by hand).
In order to create vertices of the same shape we give a cost to every vertex type: For critical vertices this cost function prefers vertices being free to the left and increases monotonically for vertices being free to the bottom, to the top and to the right (in this order). It is important that the cheapest and the most expensive type are free in opposite directions because having vertices of both types makes insertions with many bends possible. Similarly the cost function for semi-critical vertices increases monotonically from vertices being free to the left and to the bottom via the other shapes until the most expensive vertices being free to the right and to the top. Note that these costs are favourable compared with a cost function where vertices being free e.g. to the bottom and to the right are cheap because such vertices could become expensive critical vertices with the free direction to the right after future insertions, whereas the cheapest semi-

critical vertex of our cost function always becomes a relatively cheap critical vertex in the future. This consideration indicates that the chosen cost function is optimal with respect to a worst-case analysis. For practical examples also a randomized variant could be interesting.
Using the optimizing system CPLEX [5] the LP can be solved in about half a second; the value of the objective function (i.e. the number of bends) is $\frac{92}{41} \cdot n \approx 2.24 \cdot n$ if the graph is always connected and $\frac{3333}{1204} \cdot n \approx 2.77 \cdot n$ if insertions with local degree 0 are allowed.
Using this method we get not only an upper bound but also a family of examples realizing this bound (in the case of an always connected graph this example consists of a series of 41 insertions that can be iterated and need 92 bends per iteration). So we are sure that the upper bound is tight with respect to the algorithm.

5 Placing Vertices inside of the Bounding Box

Our final method to improve on the upper bounds for cost functions is very powerful in practice and even leads to an improvement of the theoretic bounds: We allow a new vertex to be placed *inside* of the bounding box if this placement does not violate the incremental invariant.

Lemma 3. *Placing a new vertex v at the intersection of free lines of two neighbours v_1 and v_2 of v does not violate the incremental invariant.*

Proof. Since v will be placed at the intersection point of two free lines there are no other vertices on these gridlines in the used directions. Thus v cannot destroy a free direction of other vertices than v_1 and v_2. After the insertion v is free in the two directions in which v_1 and v_2 were free before the insertion, so the invariant holds for v. ◇

An insertion of a vertex having local degree two now in some cases can be done without producing a new bend, while this is impossible following the insertion rules of Section 2 and 4. Vertices 10 and 13 in Fig. 6 are examples for such insertions. The enormous practical profit of this idea is clear: Especially large graphs with many vertices having a degree of less than four define a large number of such intersection points, such that the probability that the new idea can save bends is very high. Applying the new insertion method to improve on the theoretic bound is tricky: The number of intersection points does not depend only of the number and shape of the vertices drawn so far, but also of their geometric position: If there are e.g. a critical vertex v_1 having its free direction to the left and a critical vertex v_2 having its free direction to the top, then they only define an intersection point if v_1 is placed to the right of and above v_2. These conditions cannot be expressed by constraints of a linear program.
But there are situations where two vertices *always* define an intersection point: If vertex v_1 is free to the left and to the right and vertex v_2 is free to the top and to the bottom, the corresponding free lines always intersect independently

of the placement of the vertices. Thus many insertions using two such vertices as neighbours get cheaper when placing the new vertex inside of the bounding box.

Our algorithm tries to create semi-critical vertices having their free directions at opposite sides whenever possible, in order to increase the number of intersection points. For example the insertion of a vertex v of local degree one can always create a semi-critical vertex of the desired shape if the neighbour of v has degree one. Thus the cost function introduced in Section 4 must be slightly modified for semi-critical vertices. Of course vertices of degree one often define intersection points, too.

The number of variables of the LP increases considerably because the reciprocal geometric position of the vertices now plays a role: The placement of the intersection points determines the shape of the new vertices (see Fig. 4 for an example). After the insertion the vertices in question are free to the left, to the bottom and (semi-critical) to the top and right (left drawing) or to the top, to the right and (semi-critical) to the left and bottom (right drawing).

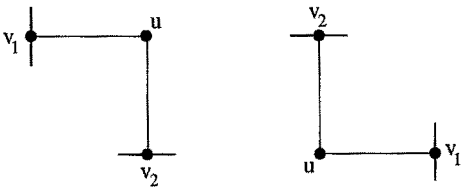

Fig. 4. Placing a new vertex u inside of the bounding box creates different situations.

The large number of cases (especially when the new vertex has local degree four) makes a complete analysis of the problem extraordenarily difficult. Inserting only the necessary rules by hand (i.e. the rules concerning types of insertions that arise in worst examples of the always-connected case) leads to an LP with 3119 variables (without ld0-insertions).

Theorem 2. *Every 4-graph without ld0-insertions allows an incremental orthogonal drawing where the number of bends is bounded by* $\frac{100}{47} \cdot n \approx 2.22 \cdot n$ *at any time, n being the actual number of vertices. Every insertion can be done in constant time.*

Proof. The value of the bound is the solution of the new Linear Program. The time bound follows from the fact that the number of intersection points is bounded by 12: There are at most four vertical gridlines that can define intersection points (the x-coordinates of the at most four neighbours of the new vertex), and on each of them there can arise at most three intersection points (having the y-coordinates of the other three neighbours). Thus the algorithm must check at most $20 + 12 = 32$ positions where the new vertex can be placed (see [9] for an exact case analysis). ◇

When allowing ld0-insertions the number of cases is so high that we did not succeed in establishing a corresponding LP by hand. We conjecture that the insertions of isolated vertices are always cheap such that the value of the objective function should be very close to the case of an always connected graph.

6 Bounds for the Used Area

We can bound the area used by our drawings using a method similar to the bend minimizing strategy: The criterion where to place a new vertex now is not the number of new bends but the number of new gridlines. We bound the total number of gridlines used by the drawing and argue that a drawing using k gridlines cannot require more than $(\frac{k}{2})^2$ area. This is because the bounding box of the drawing has a circumference of $2 \cdot k$ and the rectangle with the largest area having a circumference of $2 \cdot k$ is the square with a side length of $\frac{k}{2}$.
In our LP we only have to change the objective function (which now must count the number of new gridlines instead of the number of new bends) and get a result of $0.937 \cdot n^2$ resp. $1.057 \cdot n^2$ (allowing ld0-insertions or not), whereas Papakostas and Tollis only can guarantee an area of at most $1.77 \cdot n^2$. The profit of the idea described in Section 5 for the used area is obvious: a new vertex placed inside of the bounding box does not require new gridlines because it uses old ones. For the area it is not surprising that insertions of vertices with local degree 0 increase the upper bound because such an insertion always needs two new gridlines in order to guarantee four free directions of the new vertex, whereas an insertion of a vertex of local degree 1 requires only one new gridline. Again the bounds are tight for the algorithm.
In Section 7 we present the results on the area when using the different approaches desribed in this paper.

7 Lower Bounds and Offline Bounds

A classification of our results requires the comparison with existing results about upper bounds for the offline-version of the corresponding problems as well as the comparison with lower bounds. Table 2 shows some results on the number of bends and on the used area:

Table 2

	Bends	Area
Upper bound offline	$1.9 \cdot n$	$0.8 \cdot n^2$
Lower bound offline	$1.83 \cdot n$	$\Omega(n^2)$
Papakostas and Tollis	$2.66 \cdot n$	$1.77 \cdot n^2$
Basic algorithm	$2.5 \cdot n$	$1.44 \cdot n^2$
Section 4 always connected	$2.24 \cdot n$	$0.937 \cdot n^2$
Section 5 always connected	$2.22 \cdot n$	$0.937 \cdot n^2$
Section 4 general	$2.77 \cdot n$	$1.057 \cdot n^2$
Section 5 general	??	??

The lower bound for the number of bends given in Table 2 is a lower bound for offline drawings and is established in [1]. For the area no good lower bound is known. In [15] a lower bound of $\Omega(n^2)$ is proved and in [2] the author precises the constant of this bound to be about 10^{-6}. Thus this bound is not very useful for comparisons. The offline upper bounds in Table 2 are established in [14]. It is worth to mention that the minimization problems for the number of bends and for the used area are both $\mathcal{NP}$-complete [8,10].
The upper bounds for the area and the number of bends in incremental drawings are not much worse than the corresponding offline-bound. In [7] we prove a lower bound of $2 \cdot n$ for the number of bends in incremental drawings. This result makes the upper bound of $2.22 \cdot n$ even more valuable.
For the last line of the table the exact values are not known, but we conjecture that they are very close to the always-connected case.

8 Examples and Practical Results

To estimate the practical significance of the algorithm we compared some drawings with
a) drawings got by the algorithm in [13];
b) drawings got by an offline-algorithm.
Having a closer look at the results in [13] and [12] it can be seen that the No-Change-Scenario was mainly developped in order to apply an easy and effective technique of analysis, whereas the Relative-Coordinate-Scenario was developped for practical applications. This is the reason for the bad upper bounds of the Relative-Coordinates-algorithm ($3 \cdot n$ for the number of bends, $2.25 \cdot n^2$ for the area) compared to the weaker model. Thus a comparison of our drawings with No-Change-drawings of Papakostas and Tollis would not be fair. But our drawings are even better than the Relative-Coordinates-drawings of Papakostas and Tollis in most cases, although we guarantee the rules of the No-Change-Scenario. Fig. 5 and 6 show three drawings of the same graph produced by the three algorithms. The No-Change-drawing needs 16 bends and 13 × 13 area, the Relative-Coordinates-drawing 12 bends and 11 × 11 area and our algorithm produces only 9 bends on an area of 9 × 10. Further tests have shown that these numbers show a typical relation between the three algorithms.

9 Extensions

The algorithms described so far are all implemented in our system [9]. The methods are very flexible such that it is easily possible to think about extensions in order to have algorithms that can be used in a large field of applications.
In [7] a detailed discussion about possible extensions can be found. The goals are

- to provide interactive drawings, i.e. also deletions of vertices and edges;
- to optimize other cost functions;

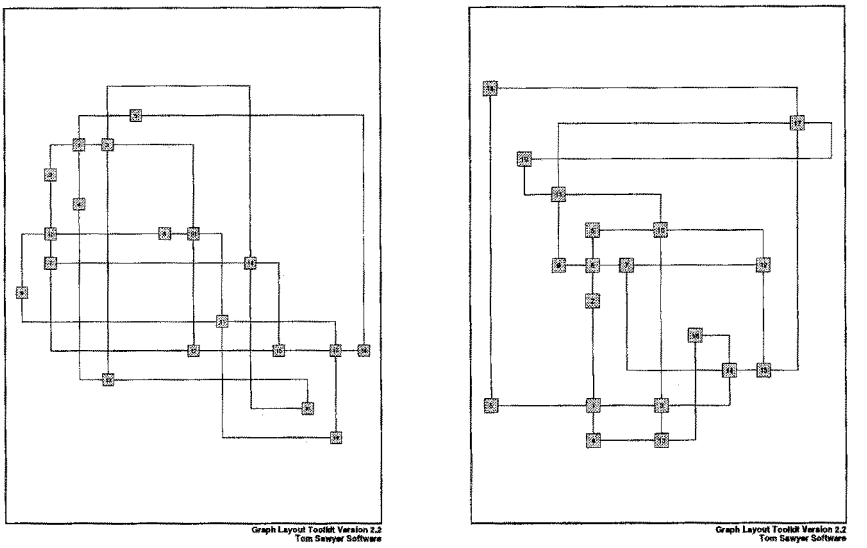

Fig. 5. No-Change Papakostas/Tollis and Relative-Coordinates Papakostas/Tollis

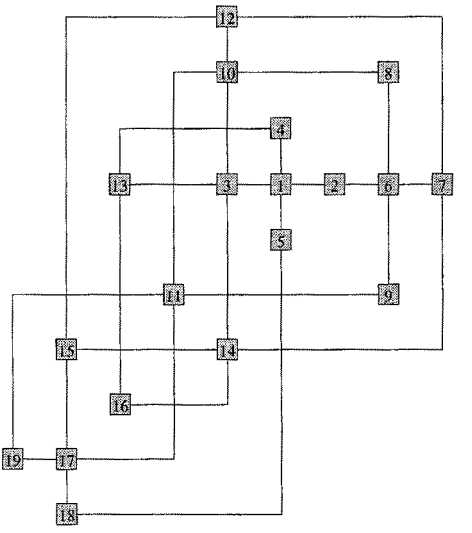

Fig. 6. No-Change new approach

- to realize the ideas of Relative-Coordinates;
- to draw graphs with an arbitrary vertex degree.

Especially the last point is important in order to have an algorithm that is universally applicable to arbitrary graphs.

Acknowledgement: I wish to thank Michael Kaufmann and Frank Jahrmarkt for useful discussions, Frank Jahrmarkt for the implementation and Achilleas Papakostas, Janet Six and Ioannis Tollis for getting some drawings of their algorithms for comparisons.

References

1. Biedl, T.C., *New Lower Bounds for Orthogonal Graph Drawings*, Proceedings on GD'95, Passau, 28-39, 1995.
2. Biedl, T.C., *Orthogonal Graph Drawing, Algorithms and Lower Bounds*, Diploma Thesis TU Berlin, 1995.
3. Batini, C., E. Nardelli and R. Tamassia, *A Layout Algorithm for Data-Flow Diagrams*, IEEE Trans. on Software Engineering, Vol. SE-12 (4), 538-546, 1986.
4. Batini, C., M. Talamo and R. Tamassia, *Computer Aided Layout of Entity-Relationship Diagrams*, The Journal of Systems and Software, Vol. 4, 163-173, 1984.
5. CPLEX optimization, Inc. *Using the CPLEX Base System.* CPLEX Optimization, Inc., 1995.
6. DiBattista G., P. Eades, R. Tamassia and I.G. Tollis, *Algorithms for Drawing Graphs: An Annotated Bibliography*, Computational Geometry: Theory and Applications, vol. 4, no 5, 235-282, 1994.
7. Fößmeier U., *Interactive Orthogonal Graph Drawing: Algorithms and Bounds*, Technical Report WSI-97-12, University of Tübingen.
8. Garg, A., R. Tamassia, *On the computational complexity of upward and rectilinear planarity testing*, Proceedings of GD'94, Princeton, 286-297, 1994.
9. Jahrmarkt, F., *Knickminimierende Verfahren für interaktives orthogonales Graphenzeichnen*, Diplomarbeit Universität Tübingen, 1997 (in German language).
10. Kramer M.R., J. van Leeuwen, *The complexity of wire routing and finding minimum area layouts for arbitrary VLSI circuits*, Advances in Computer Research, Vol. 2: VLSI Theory, Jai Press, Reading, MA, 129-146, 1992.
11. Misue, K., P. Eades, W. Lai and K. Sugiyama, *Layout Adjustment and the Mental Map*, Journal of Visual Languages and Computing, vol. 6, 183 - 210, 1995.
12. Papakostas, A., J. M. Six, I. G. Tollis, *Experimental and Theoretical Results in Interactive Orthogonal Graph Drawing*, Proceedings on GD'96, Berkeley, 371-386, 1996.
13. Papakostas, A. and I.G. Tollis, *Issues in Interactive Orthogonal Graph Drawing*, Proceedings on GD'95, Passau, 419-430, 1995.
14. Papakostas, A. and I.G. Tollis, *Improved Algorithms and Bounds for Orthogonal Drawings*, Proceedings on GD'94, Princeton, 40-51, 1994.
15. Valiant, L. G., *Universality considerations in VLSI circuits*, IEEE Trans. Comput., C-30, 135-140, 1981.